NextCloud and Our Website Connection via Custom Develop API (MemoPalestine App)

---

* Overview

This document proposes and explains the architecture we seek to follow and method of integrating our Website with Nextcloud through a custom-developed API, MemoPalestine, that is independent of Nextcloud's core API and authentication system. The aim is to create a secure, scalable, and limited-scope interface between our public website and file management system.

---

* Registration Flow

** Laravel Side (Frontend + Backend):

1. User submit registration via our website.
2. User fills required fields, including:
    - Username (optional if token is provided).
    - Access token (optional, used to link to an existing Nextcloud account).

** Case A: Token Provided

- Laravel validates the token by querying our custom MemoPalestine API (not Nextcloud core).
- If the token is invalid → Error returned to frontend: Invalid token.
- If valid:
    - Laravel stores the token encrypted in the local DB for future secure communications (or we could store it to a different database that is more secure).
    - No need to create a new user in Nextcloud.

** Case B: Token Not Provided

- Username becomes mandatory.
- Laravel makes a call to the MemoPalestine API to create a new Nextcloud user.
- If successful:
  - API returns the new user ID and a valid access token.
  - Laravel stores this data securely.
- If creation fails → Error returned related to user creation.

** Token Generation for Existing NextCloud Users

- Existing users who want to link their Nextcloud account must visit a specific page in the MemoPalestine App inside their Nextcloud account.
- This page generates an access token to use during Laravel registration.

** Final Step: Email Verification

- After successfully registering or linking, users must verify their email address.
- Registration is complete after email confirmation.

* File Upload Flow

- Uploads are initiated from the Laravel website.
- The website retrieves the user's encrypted token, and makes requests to custom MemoPalestine upload endpoints.
- These endpoints:
  - Support resumable (chunked) uploads.
  - Mimic WebDAV Chunked uploads behavior without relying on it.
  - Automatically tag uploaded files with a special tag: `WEBSITE_PUBLIC`.
    - Ensures files are treated as public and accessible through the read API.
  - Fully bypass and not relying Nextcloud's default auth system.

* File Reading Flow

- File access is strictly limited to public files.
- Public files are identified by the tag `WEBSITE_PUBLIC`.
- If a file does not have this tag → It is treated as private → Returns `404 Not Found`.

* File Deletion Flow

- Uses a custom MemoPalestine endpoint.
- Deletion is delayed:
    - File is first marked for deletion.
    - A cron job periodically deletes files marked after a specific duration.
    - This adds a safety window to undo unauthorized deletions.
- Deletion is only applicable to public files (`WEBSITE_PUBLIC` tag).
    - Private files (no tag) → `404 Not Found`.
- Suggestion: if X files requested to be deleted in less than Y duration we can revoke token and cancel the newly delete requests (as the request marked as unauthorized)

* Authentication and Security Model

The MemoPalestine API uses a completely separate auth mechanism from Nextcloud core.

** Key Security Features:

- No dependency on Nextcloud's native auth.
- Tokens:
    - Are scoped only to our custom endpoints.
    - Cannot be used to access general Nextcloud APIs.
- IP-restricted access to endpoints.
- Multi-header auth requirement:
    - `X-API-Key`
    - `X-Timestamp`
    - `X-Nonce`
    - `X-Signature`

These headers must be generated securely on the Laravel side to ensure integrity and prevent replay attacks.

* Existing Custom API Endpoints

Endpoint	Description
`/generate-token`	Web-only (not API) — for existing NextCloud users to generate their access token
`/register-user`	Creates new NextCloud user via API
`/upload/chunk`	Uploads file in chunks, mimics WebDAV Chunked upload
`/upload/assemble`	Combines chunks into final file, applies tags
`/file/meta`	Returns public file metadata (duration, dimensions, etc.)
`/tags/(sync|add|remove)`	Syncs/Add/Remove tags on a file (excluding `WEBSITE_PUBLIC`)
`/delete/mark`	Marks a public file for future deletion

> **>> NOTE:** All endpoints are protected with custom headers and scoped access and only have access to public files marked with `WEBSITE_PUBLIC`.

* Suggestions:

- Token Rotation: we could check if the token was created before 12 hours ago, and request a new token using it, with this the old token will be revoked and the new token will be stored, this can be done using Website do this before do any request, or from NextCloud requesting a Website endpoint (webhook) to update specific user token. Both ways have pros, and cons.
- Store tokens in a separated database still encrypted of course.

* Option 1 Risks:

- No TUS Protocol support for resumable uploads (No good UI for it)
- Manual account creation bottleneck.
- Mandatory tagging structure is difficult to enforce due to the flexible nature of tags assigning.
- very limited integration with the website.
- losing access to file meta data such as width, height, duration, and could need more later but even these 3 are required for website media information display.
- ssh connection and code needed to be implemented on Nextcloud side for automating publishing the file to web server and storing the web server credentials stored in NextCloud info (the full access to server??)
- Any dependency on NextCloud Custom App Development requires 24 hours for updating the modifications due to nature NextCloud servers were built on the Host.
- Using Rsync with SSH does not necessarily hide the NextCloud server IP, it is probably still known or logged somewhere on the Website server.
- While it's not impossible, it is still harder to implement a reliable PHP interface to track the status and the progress or rsync to a web app.

Conclusion

Both the suggested options have Pros and Cons, but in my opinion both still know the NextCloud server and my proposed structure and custom API define such a better security from what currently exists in the website (i'm not saying better than option 1), but with my proposed structure we got the minimal overheads keeping the same user experience, while maintain the same existing website UI functionality.